GENIE: A Hybrid Genetic Algorithm for Feature Classification in Multi-Spectral Images

Simon Perkins, James Theiler, Steven P. Brumby, Neal R. Harvey, Reid Porter,
John J. Szymanski and Jeffrey J. Bloch

Space and Remote Sensing Sciences, Los Alamos National Laboratory,
Los Alamos, NM 87545, USA

ABSTRACT

We consider the problem of pixel-by-pixel classification of a multi-spectral image using supervised learning. Conventional supervised classification techniques such as maximiun likelihood classifcation and less conventional ones such as neural networks, typically base such classifications solely on the spectral components of each pixel. It is easy to see why: the color of a pixel provides a nice, bounded, fixed dimensional space in which these classifiers work well.

It is often the case however, that spectral information alone is not sufficient to correctly classify a pixel. Maybe spatial neighborhood information is required as well. Or maybe the raw spectral components do not themselves make for easy classification, but some arithmetic combination of them would. In either of these cases we have the problem of selecting suitable spatial, spectral or spatio-spectral features that allow the classifier to do its job well. The number of all possible such features is extremely large. How can we select a suitable subset?

We have developed GENIE, a hybrid learning system that combines a genetic algorithm that searches a space of image processing operations for a set that can produce suitable feature planes, and a more conventional classifier which uses those feature planes to output a final classification.

In this paper we show that the use of a hybrid GA provides significant advantages over using either a GA alone or more conventional classification methods alone. We present results using high-resolution IKONOS data, looking for regions of burned forest and for roads.

Keywords: Genetic algorithms, genetic programming, hybrid genetic algorithms, image feature classification, remote sensing, spatial context

1. INTRODUCTION

Modern satellite and aerial remote-sensing instruments produce ever greater quantities of image data of the Earth, at higher resolutions and with more spectral channels than ever before. Making sense of this data within a reasonable time-frame requires automated systems that can quickly and reliably interpret this data and extract information of interest to analysts.

Our particular area of concern is in performing pixel-by-pixel classifications of multi-spectral remotely-sensed images, producing overlays that show the locations of features of interest. The range of features we look for is very large, ranging from broad-area features such as forest and open water, through to man-made features such as roads and buildings. The broad range of features we're looking for, and the variety of instruments that we work with, make hand coding feature-finders impractical. Therefore we use a supervised learning scheme that, starting from a few hand-classified images, can automatically develop image processing pipelines that can distinguish the feature of interest from non-features. As an added bonus, it is often much quicker to develop a feature-detector automatically in this way, compared with writing the detector by hand, and so our system is very useful when a new feature must be located quickly in a large data set.

Supervised learning applied to remote-sensed images is not a new field, and many different statistical and machine learning techniques have been tried, ranging from maximum likelihood classifiers through to neural networks. Chapter 8 of Richards and Jia[1] provides a good summary.

The vast majority of supervised learning applied to remote-sensed imagery bases classification purely upon the feature vectors formed by the set of intensity values in each spectral channel for each pixel. This vector of numbers

E-mail contact: s.perkins@lanl.gov

provides a nice fixed-dimensionality multi-dimensional space in which conventional classifiers work well. However, it is often the case that spectral information alone is insufficient to correctly identify a pixel — often some feature of its neighborhood, e.g. texture, or the average value of nearby pixels, is necessary to disambiguate the spectral information. We can imagine that many different kinds of extra spatial context information could be added into the pixel feature vector as additional feature dimensions. Now consider that even in the spectral domain it may be that the raw spectral intensity values do not make for an easy classification. For instance, perhaps the normalized vector of spectral intensities would be easier to work with, and provide better robustness with respect to varying illumination. It is easy to see that there are a large number of choices for additional feature vector dimensions, and easy to see that the correct choice could make classification much easier than just taking the raw spectral values as our feature vector. The question is: how do we choose a suitable set of features automatically?

We have developed a hybrid evolutionary algorithm called GENIE to do just this. The essential idea behind GENIE is that a genetic algorithm[2] searches a very large space of image processing algorithms that transform raw multi-spectral pixel data into a new set of image planes that we call *feature planes*. A conventional supervised classifier is then applied to these feature planes and outputs the final classification. It is our experience that this hybrid combination of evolutionary search and conventional classification techniques can be very powerful.

Previous work[3–5] has described some of the empirical results achieved with GENIE. This paper concentrates on the hybrid aspects of the system, and in particular compares the performance of the hybrid system with a pure evolutionary system and with a pure conventional classifier.

2. THE GENIE SYSTEM

2.1. Training Data

In GENIE, we typically assume that training data is provided by a human expert analyst, marking up an image by hand, showing both locations of the feature of interest, and locations where that feature is definitely not found. We have developed a Java-based tool called ALADDIN which tries to make this process as painless as possible. ALADDIN provides a 'paint program' environment in which an analyst can simultaneously view a false color projection of a multi-spectral image, and a grayscale version of the same image on which he or she can 'paint' training data as a colored overlay. ALADDIN provides many user-friendly features to make this job easier, such as the ability to zoom into an image to markup fine detail, and the ability to map multi-spectral images into RGB space in a large number of ways. A key point to note is that we do not require every pixel in an image to be classified by the analyst. In fact typically the analyst will only markup small portions of a large image, illustrating the various different contexts in which the desired feature appears, and similarly a variety of contexts in which the feature does not appear. Only the regions specifically marked up will be used as training data. The user can further restrict the amount of training data by drawing rectangular bounding boxes around selected regions of interest. Using ALADDIN several different images can be marked up with the same feature, and these will all be used as training data. Figure 1 shows a screenshot of the user interface. Similar interfaces have been used to create training data for evolutionary algorithms by Bersano.[6]

2.2. The Learning System

At the heart of GENIE is a genetic programming system based on a linear chromosome (see Banzhaf et al.[7] for a good introduction to GP with an emphasis on linear genomes). The GP system manipulates image processing programs that take the raw pixel data planes and transform them into a set of *feature planes*. This set of feature planes is in effect just a multi-spectral image of the same width and height as the input image, but perhaps having a different number of planes, and derived from the original image via a certain sequence of image processing operations. GENIE then applies a conventional supervised classification algorithm to the feature planes to produce a final output image plane, which specifies for each pixel in the image, whether that feature is there or not. A fitness for the image processing program just examined is then calculated by comparing this output image with the training data in a manner described below. Figure 2 illustrates this hybrid scheme.

2.2.1. Genetic Programming Details

GENIE uses a fixed-length linear chromosome, rather than the more conventional tree-based representation typically used in GP, and we use standard one-point crossover, rather than some form of more sophisticated sub-tree or sub-graph crossover. This choice is motivated partly by a desire to produce code similar to that which a human might produce (the individual elements in our genome actually correspond directly to lines of code written in the

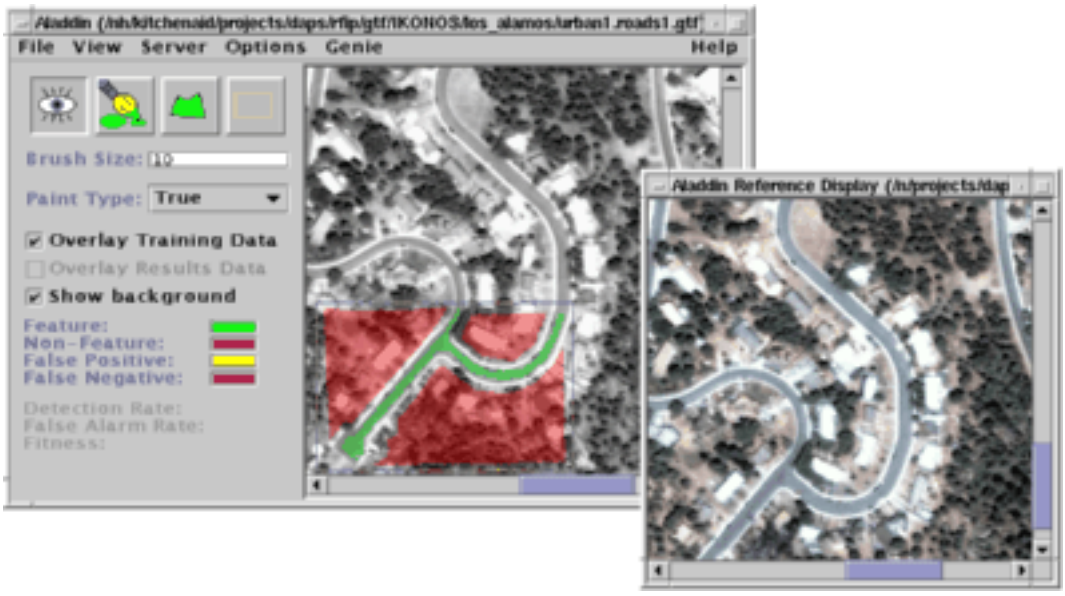

Figure 1. The ALADDIN interface. Training data is painted onto a grayscale version of an image as a colored overlay. In this example, roads inside a user-specified bounding box have been marked up in green to indicate that this feature is to be found, while the surrounding area has been marked up in red. This distinction will not be obvious if you're reading a black and white version of this paper!

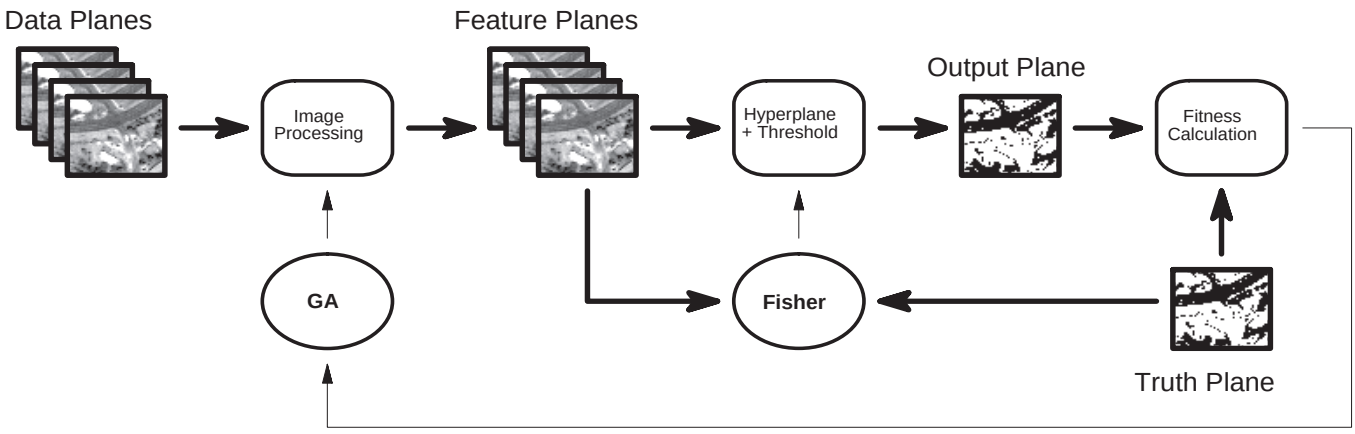

Figure 2. The overall structure of GENIE. Raw data planes are transformed into a set of feature planes by an image processing program that is evolved by the GA. A discriminating hyperplane is then applied to those feature planes to produce a final classification. The hyperplane is found using a combination of Fisher discriminant and a threshold search. The output classification is compared with the user-supplied truth to obtain a fitness score, which is passed back to the GA.

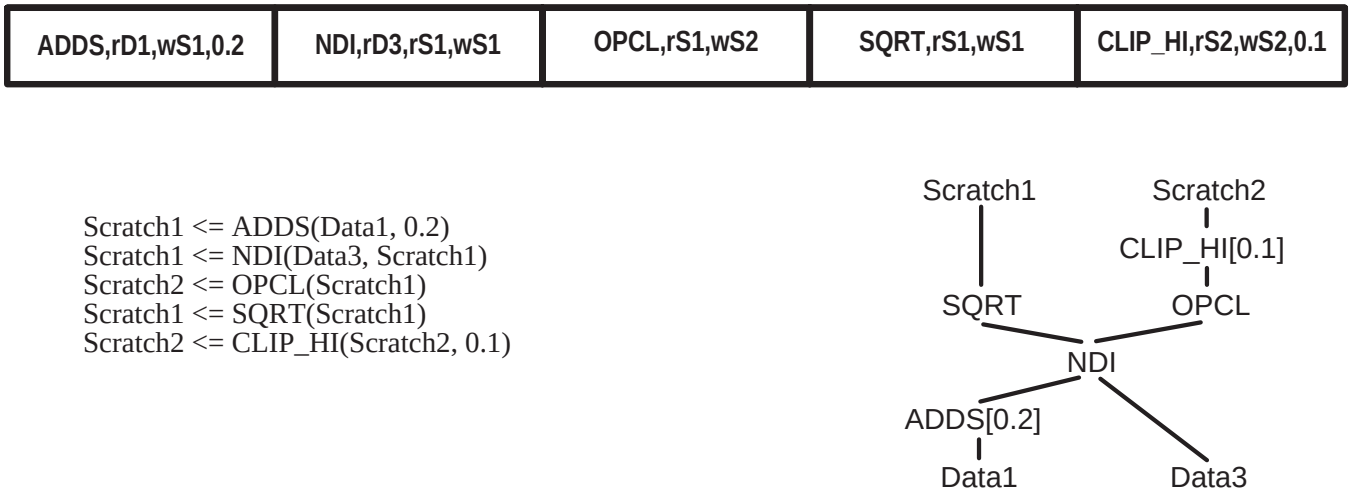

Figure 3. Three equivalent views of a chromosome. At the top is the raw linear genome representation that we use. Lower left shows a 'lines-of-code' version of the genome. This is the form closest to what gets executed. Lower right is a graph representation of the same genome. Note that the genome representation allows more complicated graphs than just simple trees.

data processing language IDL), and partly by the observation that it seems to work well. It is interesting to note that several others in the genetic programming community, e.g. Francone et al.,[8] have claimed significantly better performance with simple 'homologous' crossover on fixed length strings, compared with more flexible crossover schemes.

A single chromosome is made up of a string of genes, each one of which corresponds to a particular image processing operation. Each gene has one or more inputs, and one or more outputs. An input can be taken from any one data plane in the original image (there are as many data planes as there are spectral channels), or from any one 'scratch plane'. Scratch planes are temporary holding places where a single image plane can be held. The gene performs some image processing operation on its inputs and produces one or more planes of output data. Each of these planes is written to a different scratch plane. The whole chromosome is evaluated by starting with the gene at the left end, and sequentially stepping through the genes in order, one-by-one. It is a requirement when chromosomes are created that no gene is allowed to read from a scratch plane that has not been written to by at least one gene to the left of it. In addition, in these experiments, we impose a requirement that all scratch planes must be written to at least once during a chromosome's execution. The feature planes which are passed on to the backend classifier, are specified by the user as a subset of the scratch and data planes. Figure 3 shows how our genome representation is translated into an image processing pipeline.

Note that it is quite possible that a gene will write its output to a scratch plane, which is then overwritten by another gene before it is ever used. In this case that gene is irrelevant, as are any genes that write data to scratch planes that are only read by irrelevant genes before being overwritten. Hence, although the chromosome length is fixed, the effective program length can vary significantly. In GENIE, we perform an efficient graph analysis of the chromosome and determine which genes are irrelevant. Those genes are kept in the chromosome, but for efficiency, are never actually executed.

Each gene corresponds to a different image processing operation, but the details of that operation can be influenced by gene parameters. Different genes have different numbers of parameters, and each parameter is associated with a fixed set of attributes that determine such things as what range it is randomly initialized within when that gene is first created, what range of values it can possibly take, and how it is affected by mutation (see below). We have three kinds of parameters: (*i*) float parameters are initialized to a random floating point number in the initialization range, and are mutated by a floating point offset that is Gaussian distributed with a standard deviation given by that parameter's *delta* attribute; (*ii*) integer parameters are initialized to a random integer in the initialization range, and are mutated by an integer offset that is uniformly distributed in a range given by plus or minus the *delta* attribute; (*iii*) symbolic parameters are like integers, but when mutated are simply re-initialized randomly. In general, genes

attempt to produce output that is roughly on the order of the same scale as their input. A variable called `dataScale` is set to the range of values found in the training data, and genes that naturally have an output that is of a different range to their input, rescale their output to be of the order of `dataScale`. Table 1 lists all the genes used in our runs.

2.2.2. Conventional Classification Details

The image processing program specified by the genome transforms raw data planes into a set of feature planes. Typically, we take the scratch planes to be the feature planes. A conventional classification algorithm is then used to produce the final output classification, on which the genome fitness is based. The classifier uses the training data provided by the analyst, and attempts to find the best classifier it can that reproduces that training data. Since every genome evaluation requires a pass of the backend classifier, we require that this classifier run very quickly. As a result we use the relatively simple *Fisher linear discriminant*. See, for example, Bishop[9] for a detailed description of this algorithm, but put simply the Fisher discriminant is an optimal (in some sense) linear discriminant in feature space that takes into account the means of the true and false pixel clusters as well as their covariance matrices. The computational requirement for the Fisher discriminant is $\mathcal{O}(n^2)$ in the number of pixels,* and $\mathcal{O}(n^3)$ in the number of feature planes. In fact the Fisher discriminant only gives the normal to the discriminant hyperplane — we then do a line search ($\mathcal{O}(n)$ in the number of pixels) to find the optimal threshold along that direction. The output from the classifier is a single image plane containing 1's and 0's corresponding to true and false.

2.2.3. Fitness Calculation

Our fitness measure assigns equal importance to getting all the pixels marked as true correct, and getting all the pixels marked as false correct. The *detection rate* R_d is defined as the fraction of pixels marked as true that the classifier got correct, and the *false alarm rate* R_f is defined as the percentage of pixels marked as false that the classifier got wrong. The fitness F is then defined as:

$$F = 500 \times (R_d + (1 - R_f))$$

which gives a number between 0 (bad) and 1000 (perfect). In fact due to the operation of the Fisher discriminant, it is impossible to get a score less than 500 on the training data. The fitness score is then assigned to the genome that was just used.

When the initial population of chromosomes has been evaluated, conventional tournament selection is used to select parents for the next generation. Two parents are selected at a time. Single point crossover is performed with probability P_c. We perform up to N_g single-gene mutations on each individual, where N_g is the number of genes in the genome. Each of these mutations has a $\frac{P_m}{N_g}$ probability of happening, but if we didn't perform crossover, then at least one mutation always happens. Three kinds of mutation can happen: replacement of the gene with a new random gene, mutation of a single parameter of a gene, and mutation of a single input or output plane specification. These mutations happen in the ratio 2:1:1. Elitism is employed so that the single best individual is copied into the next generation. A check is performed on offspring chromosomes to see that they do not contain genes that read from uninitialized scratch planes, and to make sure that all the scratch planes are used at least once. Offspring that fail this check are discarded. Breeding continues until a new population of equal size to the original population has been created, and the cycle repeats.

2.3. Examining Results

At the end of a run, the best classifier genome is saved, along with the coefficients describing the backend classifier hyperplane. The analyst can then run this algorithm over other images and see whether the classifier is doing a good job. ALADDIN is again used as a tool for overlaying classification results on top of an image. If additional marked up images are available, then ALADDIN can overlay results data on top of training data to show where the evolved algorithm disagrees. On the basis of this the analyst might decide to mark up additional training data, and begin a new training run, perhaps seeding the run with the best of the previous run. This iterative process continues until the analyst is satisfied that a robust classifier has been developed. We are currently also examining other ways of combining together classifiers evolved during different runs using techniques such as *boosting*.[10]

*Note that only pixels in the images that have been assigned true or false values are considered.

Gene Abbreviation	Image Processing Operation	Inputs/ Outputs/ Params	Notes
ADDP	Add planes	2/1/0	Basic mathematical operations. ADDS adds a scalar, which may be negative, to its input. DIFF is like SUBP but outputs the absolute values. NDI is like SUBP, but divides the result by the sum of its two inputs. MULTS scales its input by a scalar, which by default is positive. LINSCL is like MULTS but takes an extra param which is added onto the scaled input. LINCOMB outputs a linear combination of its two inputs, in proportion specified by its one parameter, which takes a value between 0 and 1.
ADDS	Add scalar	1/1/1	
SUBP	Subtract planes	2/1/0	
DIFF	Absolute difference	2/1/0	
NDI	Normalized difference	2/1/0	
MULTS	Multiply by scalar	1/1/1	
NEG	Negate plane	1/1/0	
MULTP	Multiply planes	2/1/0	
SQRT	Square root	1/1/0	
SQR	Square	1/1/0	
LINSCL	Linear scale	1/1/2	
LINCOMB	Linear combination	2/1/1	
MIN	Minimum	2/1/0	Logical operations. MIN and MAX perform pixelwise minimum and maximum, equivalent to AND and OR for binary input. IFLTE outputs its third input wherever the first input is less than its second input, and its fourth output elsewhere.
MAX	Maximum	2/1/0	
IFLTE	'If less than else'	4/1/0	
CLIP_HI	Clip high	1/1/1	Thresholding operations. CLIP_HI truncates any pixel values above a value set by its param. CLIP_LO does the converse. THRESH sets all values below its threshold param to 0, and all those above to `dataScale`.
CLIP_LO	Clip low	1/1/1	
THRESH	Threshold	1/1/1	
SAVAR	Spectral angle variance	2-16/1/2	Spectral angle operations. SAVAR and SADIST look at two circular neighborhood regions around each pixel, of size defined by their two params. SAVAR returns the difference between the variance of the spectral angles of the pixels in the two regions. SADIF returns the difference between the mean spectral angle of both regions. SADIST returns the spectral angle difference between each pixel and the vector defined by its params. SANORM normalizes the vector defined by its inputs to have a magnitude equal to `dataScale`.
SADIF	Spectral angle difference	2-16/1/2	
SADIST	Spectral angle distance	2-10/1/2-10	
SANORM	Normalize spectral vector	2-10/2-10/0	
R5R5	Lawe's texture measure	1/1/0	Neighborhood operations. In general, all these operations take a single plane as input and produce a single output plane. The output at each pixel is determined by looking at the pixel's neighborhood. R5R5, LAWB, LAWD, LAWF and LAWH are widely-used texture measures, developed by Lawe, that return zero if the neighborhood contains all the same value of pixel, and some other value otherwise, depending upon the distribution of pixel values. R5R5 is corresponds to Lawe's $R5^T \times R5$ 5×5 operator. The others are 3×3 operators, corresponding to Lawe's $S3^T \times L3$, $E3^T \times E3$, $L3^T \times S3$ and $S3^T \times S3$ operators repsectively. Most of the other operators are familiar image processing or morphological operators, whose description can be found in any good book on image processing. Most take a single param which gives the radius of a circular structuring element defining the neighborhood to which the operator is applied. ASF stands for 'Alternating Sequential Filter'. MB_EDGE takes an additional parameter defining a threshold for edge strength to be looked for. The single parameter for H_DOME and H_BASIN defines the pixel value offset used by these operators.
LAWB	Lawe's texture measure	1/1/0	
LAWD	Lawe's texture measure	1/1/0	
LAWF	Lawe's texture measure	1/1/0	
LAWH	Lawe's texture measure	1/1/0	
LAPLAC3	3x3 Laplacian	1/1/0	
LAPLAC5	5x5 Laplacian	1/1/0	
MORPH_LAPLAC	Morph. Laplacian	1/1/1	
ISO_GRAD	Isotropic gradient	1/1/0	
MEAN	Mean	1/1/1	
VARIANCE	Variance	1/1/1	
SKEWNESS	Skewness	1/1/1	
KURTOSIS	Kurtosis	1/1/1	
SKEW_COEFF	Skewness coefficient	1/1/1	
KURT_COEFF	Kurtosis coefficient	1/1/1	
SD	Standard deviation	1/1/1	
RANGE	Range	1/1/1	
MEDIAN	Median	1/1/1	
EROD	Erode	1/1/1	
DIL	Dilate	1/1/1	
OPEN	Open	1/1/1	
CLOS	Close	1/1/1	
OPCL	Open-close	1/1/1	
CLOP	Close-open	1/1/1	
ASF_CLOP	ASF Close-open	1/1/1	
ASF_OPCL	ASF Open-close	1/1/1	
POS_TH	Positive top hat	1/1/1	
NEG_TH	Negative top hat	1/1/1	
OPEN_REC	Open with reconstruction	1/1/1	
CLOSE_REC	Close with reconstruction	1/1/1	
H_DOME	H-dome	1/1/1	
H_BASIN	H-basin	1/1/1	
MB_EDGE	Canny edge detector	1/1/2	

Table 1. The genes used in GENIE and what they do. Unfortunately, space precludes a complete description of the details of the more complex operators, but this table gives the general picture of what kinds of operators are in GENIE.

3. EXPERIMENTS

GENIE is a hybrid learning system. The aim of the experiments in this paper is to compare the performance of the hybrid system, with each of the individual learning components taken separately. In doing so we will also illustrate GENIE finding successful solutions to two interesting feature classification problems.

We used data from the recently launched IKONOS satellite.[11] This satellite produces commercially available imagery of the Earth at very high resolution. The raw data product consists of panchromatic images at 1 m resolution, and 4-color (blue, green, red and near-IR) images at 4 m resolution. We used 4-color imagery that has been 'sharpened' to 1 m resolution using the panchromatic image — the intensity of the high-resolution panchromatic image is used for each pixel, with chromaticity provided by the low-resolution color image. The images used in these experiments consist of 1000×1000 tiles cut from a large 10928×10928 IKONOS image of Los Alamos County in New Mexico, USA. The IKONOS data provides particularly interesting tasks for GENIE since the high resolution shows up considerable texture, meaning that a simple color-based classifier often performs badly.

GENIE was tested on its abilities to find two different kinds of features: (A) roads and large paved areas, and (B) regions of forest that suffered severe burn damage during the Cerro Grande forest fire of May 2000. Two 1000×1000 tiles were created for each feature type, and partially marked up with the appropriate features using ALADDIN. Figure 4 shows the images and the associated truth markup. Images A1 and B1 were used as training data for GENIE runs, and the resulting algorithms were then tested on images A2 and B2.

Four experiments were performed using both training sets:

1. **GA Only:** GENIE was run with the backend disabled. The image processing part was run in the same way as described before, but then the contents of the first feature plane, thresholded at a value of 128 (half the pixel value range in the raw image) was used directly as the classification output.
2. **GA+Threshold:** As the GA Only run, but instead of the fixed threshold, an optimal threshold is found using a 1-D search of values in the first feature plane.
3. **GA+Fisher:** The full GENIE system as described above, including the Fisher Discriminant backend and optimal thresholding.
4. **Fisher Only:** Disabling the evolutionary component this time, and simply passing the four input data planes through directly to the feature planes, before deriving the Fisher Discriminant and optimal threshold.

The idea behind these experiments is to examine the relative importance of the evolutionary and backend components of GENIE and to demonstrate how they work together.

The following parameters were used in the GA: population size = 100, $N_g = 20$, number of scratch planes = 4, tournament size = 2, elitism = 1, $P_c = 0.9$ and $P_m = 0.5$. The feature planes were specified to be all the scratch planes. This set of parameters was chosen from experience, and has been found to work fairly well over many different runs of GENIE. Each run was continued until 5000 functionally different chromosomes had been evaluated.[†] Each experiment was repeated 5 times with different random number seeds and the results are an average of the 5 runs.

4. RESULTS

Figure 5 shows the fitness of the best individual *vs.* number of functionally distinct chromosomes evaluated, for each of the three experiments involving an evolutionary component, and for each of the two training sets used. Table 2 summarizes the best final performance for all four experiments on both training and test sets. Note that these 'best' scores are an average of the best from 5 separate runs of each experiment.

The graphs in Figure 5 show that the GA+Fisher runs attain a higher fitness, faster (in terms of chromosome evaluations) than either the GA Only or the GA+Threshold runs. Table 2 shows the full GENIE runs generally attaining a higher final level of performance than any of the other runs on both training and test data, although given the small sample size, the picture is not completely conclusive.

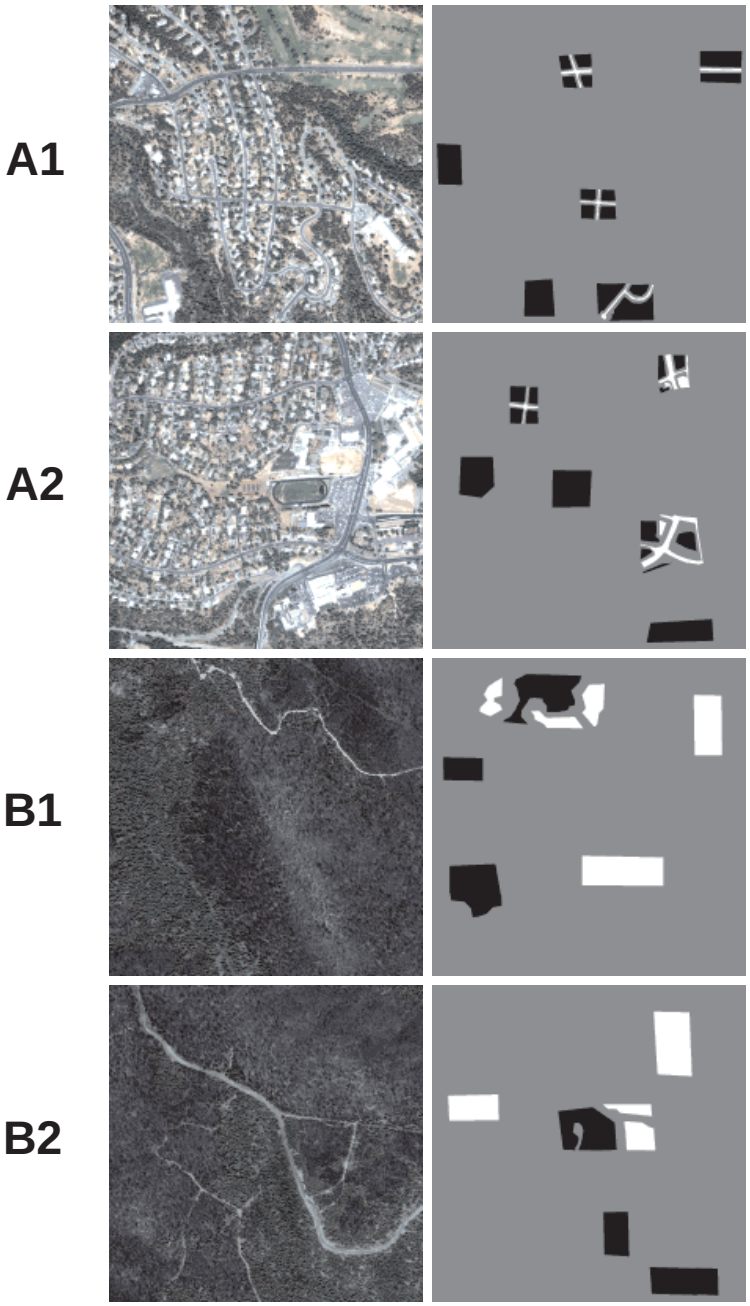

Figure 4. Training and test data used for GENIE runs. The left hand column shows the 1000×1000 IKONOS tiles, with just the visible bands shown. The right hand column shows the training/test data that has been marked up for each tile. White indicates where the analyst has marked the feature of interest as being present, black indicates places that have been marked as not containing the feature, and gray indicates places that have not been marked. As can be seen, only small portions of each image have been marked up, which reduces the amount of image processing GENIE must perform on each image.

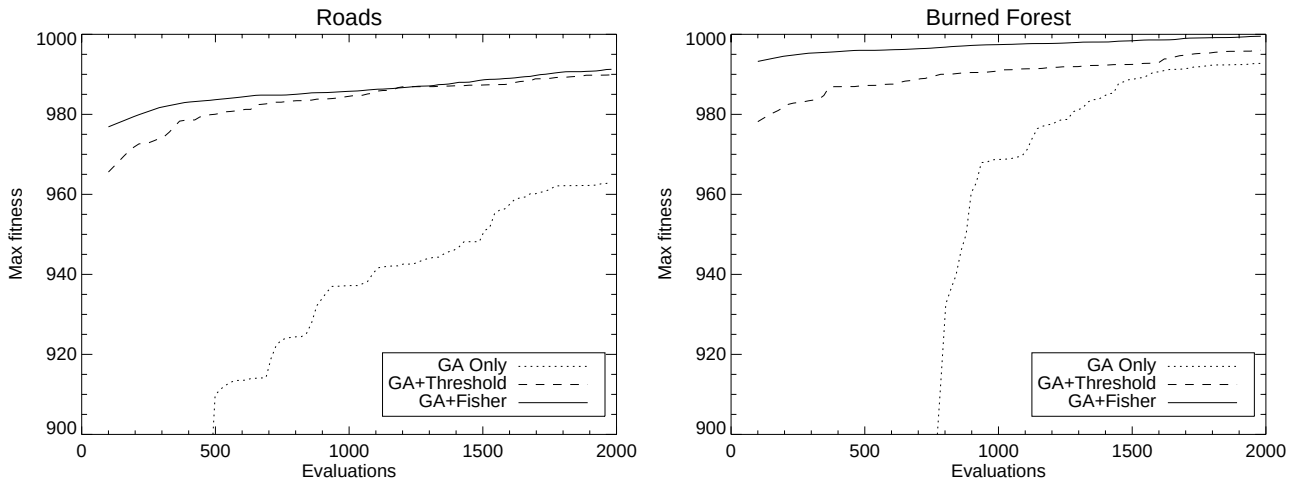

Figure 5. Plots of maximum fitness (averaged over 5 runs) *vs.* number of chromosome evaluations, for each of the three evolutionary experiments, and for each training set.

	Roads		Burned Forest	
	Training	Test	Training	Test
GA Only	965.2 ± 6.8	944.3 ± 9.5	991.6 ± 2.4	964.9 ± 1.6
GA+Threshold	987.6 ± 1.5	972.0 ± 2.6	997.5 ± 1.3	946.7 ± 7.8
GA+Fisher	988.7 ± 0.5	965.2 ± 4.5	999.0 ± 0.57	978.7 ± 6.8
Fisher Only	972.9	963.2	981.7	968.5

Table 2. Final maximum training and best test fitnesses (averaged over 5 runs) for all experiments and both training/test sets. The uncertainty in the fitness of the evolutionary runs gives the standard error of the mean of the 5 runs.[§]The Fisher Only experiment is deterministic and so has no uncertainty.

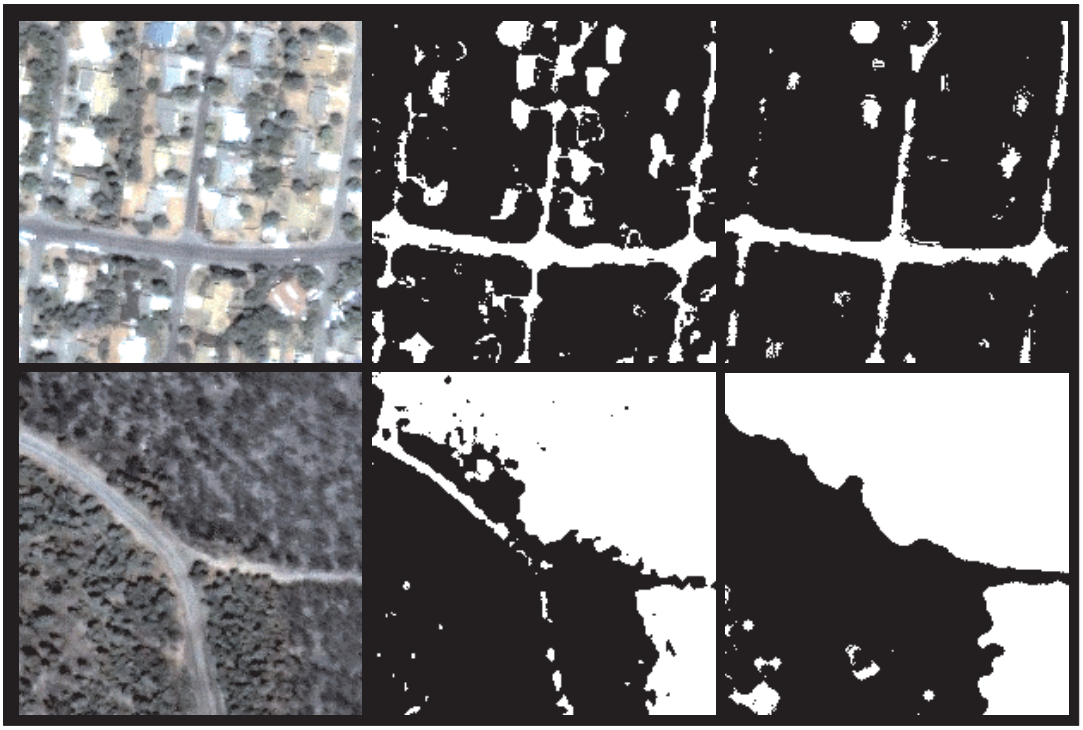

Figure 6. These images compare the classification output of the full GENIE and Fisher Only classifiers on a small part of the IKONOS test images. On the left are small portions of the IKONOS images A2 (top) and B2 (bottom). The middle column shows the classification resulting from just applying the Fisher discriminant, while the right hand column shows the GENIE classification. Neither algorithm does perfectly, but it is easy to see that the purely spectral Fisher Only classification is unable to distinguish between, say, roads and house roofs of similar color. The GENIE classifier on the other hand is able to use spatial context to disambiguate the two largely.

It is interesting to see how classifications produced by the Fisher Only experiments compare subjectively to those produced by the full GA+Fisher runs. Figure 6 shows such a comparison.

Limited space precludes a detailed analysis of the feature planes that GENIE came up with for these tasks, but one or two basic things are apparent. For the burned forest problem, many of the successful genomes produced a feature plane that was some sort of smoothed version of the near-IR channel. Living vegetation is generally brighter than the severely burned areas, and smoothing helps fill in the remaining textural variation. For the road finder the nature of the feature planes that GENIE came up with is more obscure.

Final performance is not the whole issue of course. Since every single evaluation of a genome involves computing a Fisher discriminant, the Fisher Only experiments are thousands of times faster to run than the full GENIE experiments: several seconds to get an answer compared to a small number of hours for GENIE. The time taken to perform the image processing is of a similar order of time to that taken to compute the discriminant, and so the GA Only Runs take a similarly long time to run — although they are faster than the full GENIE runs. Clearly there is a tradeoff between final performance and speed. In the typical scenarios that we work with, we can afford to take a few hours to get good final performance, but for some situations this may not be the case. An important point to note is that the GA operation is extremely parallelizable, and elsewhere[12] we have reported experiments on speeding up GENIE using a cluster of workstations.

[†]We define 'functionally different' chromosomes as two chromosomes containing identical genes in the same order, after parsing to remove redundant genes.

[§]The standard error of the mean is defined as the sample standard deviation divided by the square root of the number of samples.

5. FURTHER WORK

In this paper we have described the GENIE system in considerable detail and we have presented results indicating that the combined GA plus conventional classifier system achieves higher performance than either the conventional classifier or the GA alone.

Further experiments need to be done to confirm this result conclusively. Another interesting question is whether some other kind of conventional classifier connected to the GA would work even better. We always have to bear in mind though that the more complicated the classifier, the slower each evaluation of the GA will become.

Our experience with GENIE has been that it can indeed automatically develop classification algorithms that are competitive with hand-designed classifier. We hope to continue to demonstrate this in future work, as well as working on improving the robustness and speed of the whole system.

ACKNOWLEDGMENTS

The GENIE system is the result of the combined efforts of a fairly large group of people at Los Alamos National Laboratory, including, in addition to the authors of this paper: Melanie Mitchell, Cody Young and Kevin Lacker.

REFERENCES

1. J. Richards and X. Jia, *Remote Sensing Digital Image Analysis*, Springer, New York, 3rd ed., 1999.
2. J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.
3. J. Theiler, N. Harvey, S. Brumby, J. Szymanski, S. Alferink, S.Perkins, R. Porter, and J. Bloch, "Evolving retrieval algorithms with a genetic programming scheme," in *Proc. SPIE 3753*, pp. 416–425, 1999.
4. S. Brumby, J. Theiler, S. Perkins, N. Harvey, J. Szymanski, J. Bloch, and M. Mitchell, "Investigation of feature extraction by a genetic algorithm," in *Proc. SPIE 3812*, pp. 24–31, 1999.
5. N. Harvey, S. Perkins, S. Brumby, J. Theiler, R. Porter, A. Young, A. Varghese, J. Szymanski, and J. Bloch, "Finding golf courses: The ultra high tech approach," in *Real World Applications of Evolutionary Computing*, S. C. et al., ed., vol. 1803 of *Lecture Notes in Computer Science*, Springer, 2000.
6. T. Bersano-Begey, J. Daida, J. Vesecky, and F. Ludwig, "A Java collaborative interface for genetic programming applications: Image analysis for scientific inquiry," in *Proc. IEEE Int. Conf on Evolutionary Computation*, IEEE Press, Piscataway, NJ, USA, (Indianapolis), Apr. 1997.
7. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction*, Morgan Kaufmann, San Francisco, CA, 1998.
8. F. Francone, M. Conrads, W. Banzhaf, and P. Nordin, "Homologous crossover in genetic programming," in *Proc. Genetic and Evolutionary Computation Conference (GECCO'99)*, B. et al., ed., vol. 2, pp. 1021–1026, Morgan Kaufmann, San Francisco, CA, 1999.
9. C. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.
10. Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," in *Machine Learning: Proc. 13th Int. Conf.*, pp. 148–156, Morgan Kaufmann, 1996.
11. Space Imaging Inc. website: http://www.spaceimaging.com/.
12. N. Harvey, S. Brumby, S. Perkins, R. Porter, J. Theiler, A. Young, J. Szymanski, and J. Bloch, "Parallel evolution of image processing tools for multispectral imagery," in *Proc. SPIE 4132*, (San Diego, CA), July 2000.